swap memory location into the shared memory location at block 513. At block 515 the thread indicates that it has performed the atomic operation (e.g., sets an execution bit to true).

[0048] Figure 6C is a diagram illustrating the thread 603 updating the shared memory location 607 according to one embodiment of the invention. In Figure 6C, the thread 605 performs the CAS instruction of the program unit 409 to ensure atomicity of the memory update operation. In accordance with the example of the CAS instruction described above, the thread 603 determines if the shared-value in the shared memory location 607 is equal to the compare value in the memory location 609. If the values are equal, then atomicity for the memory update operation has not been violated and the thread 603 loads the swap value from the memory location 611 into the shared memory location 607. When the thread 605 performs the exemplary CAS instruction of the program unit 409, the shared-value and the compare value are not equal because the thread 603 has updated the shared memory location 607 with the thread's 603 swap value. Hence, the thread 605 determines that atomicity has been violated from its perspective (assuming that the thread's 603 swap value is different than the shared-value originally stored in the shared memory location 607).

[0049] Figure 6D is a diagram illustrating the thread 605 performing the loop of the program unit 409 according to one embodiment of the invention. Since atomicity has not been preserved for the thread 605, the thread 605 loads the shared-value, which is the thread's 603 swap value, from the shared-memory location 607 into the memory location 615. The thread 605 will perform the callback routine again to determine its swap value and store the swap value into the memory location 617. The thread 605 will repeat this loop until the thread 605 is terminated or the thread 605 successfully performs the memory update operation atomically.

[0050] The present embodiments of the invention provide efficient, scalable, parallel atomic operations. A single instantiation of the present embodiments of the invention can implement efficient atomic operations across multiple platforms (i.e., variants of hardware architecture, operating system, threading environment, compilers and programming tools, utility software, etc.) and yet optimize performance for each individual platform. Furthermore, the present embodiments of the invention provide the ability to tailor performance and explore performance/implementation-cost trade-offs on each of multiple platforms. For example, low-level instruction sets (i.e., assembly code instructions) may be developed to support specified operators, data-types, and/or data-type sizes at a reasonable cost while still providing the ability to optimize other operators, data-types, and/or data-type sizes differently with the described embodiments of the present invention. The present embodiments of the invention provide for using low-level instruction sets to fully support a spectrum of operations, data-types, and data-type sizes on an individual platform in order to optimize performance on that platform, while still providing the ability to optimize performance separately on other platforms.

[0051] While the invention has been described in terms of several embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. In various embodiments of the invention, program units are translated from source code to source code or from source code to object code or assembly language. In alternative embodiments of the invention, an interpreter interprets program units, code is inlined instead of making calls to a runtime library, etc.

[0052] Furthermore, the atomic update to a shared memory location can be performed by a thread as described, processor, operating system process, etc.

[0053] The method and apparatus of the invention can be practiced with modification and alteration within the spirit and scope of the appended claims.

Embodiments of the invention are described with reference to an atomic operation. The form of an atomic operation depends on the implementing language. For example, an atomic operation may take the form of a directive and a memory update operation, a call to a single function defined as an atomic operation, etc. The description is thus to be regarded as illustrative instead of limiting on the invention.